

Application of symbolic computations software to checking student's input in the intelligent tutoring web system "Volga"

Smirnova N.V., Abramnikov A.N., Dushkin D.N.

Russian Academy of Sciences, Federal State Budget Institution of Science
V.A. Trapeznikov Institute of Control Sciences
Moscow, Russia
e-mail: smirnovanatalia2008@gmail.com

Abstract—The paper describes certain issues of implementation and possible limitations of students' input check procedure where SymPy (a Python library for symbolic mathematics) is used. The suggested procedure was implemented in the intelligent tutoring web system "Volga".

Keywords—symbolic computations, solver, intelligent tutoring system, computer-aided automatic assessment

I. INTRODUCTION

Most existing tutoring systems (including interactive platforms for distance learning developed by leading America's leading Universities – Coursera, Udacity, EdX) still impose strong restrictions on the form of students' input. Analysis of more advanced students' input (i.e. presented as a set of formulas) remains to be an urgent problem.

This paper describes the first version of subsystem which allows to not only determine correctness of input, but also to ascertain which predetermined solution path student is following, whether student's solution is complete and detailed enough. It extensively uses SymPy[1], a Python library for symbolic mathematics. This subsystem is being developed as a part of intelligent tutoring system (ITS) "Volga"[2].

II. RELATED WORK

Incorporation of Computer Algebra Systems (CAS) into ITS architecture allows to significantly reduce cost of development. ActiveMath[3] and STACK[4] are the most known computer-aided assessment systems that use CAS for students' input analysis. However, these systems concentrate on analyzing student-provided answers in the form of single mathematical expressions.

To our knowledge, the most known current system designed for reasonably free problem solving at the level of university courses is the ANDES Tutor [5], designed for use in a standard two-semester introductory physics course. As Andes developers note [6], "In earlier versions of Andes the correctness of student equations was judged by whether the equation was equivalent to one on a list. The list was generated during the preparatory process by applying a set of rules for algebraic manipulation to the set of basic or "canonical" equations. If the student's equation could be found as a simple algebraic manipulation of one of the derived equations on this list it was considered correct, and which equations it depended on was given by the

corresponding set. This method requires combining the full set of canonical equations in all combinations a student might generate correctly. Unfortunately the number of canonical equations involved, even in fairly simple physics problems, is much larger than a typical human solver can imagine."

Andes developers decided to construct a different subsystem (description of current subsystem used in Andes see in [6]). We believe that the problem described above can be efficiently solved (see section 4). Therefore we designed a first version of subsystem that extensively uses a CAS-like library SymPy for analysis of students' input presented as a set of formulas.

Unfortunately, unlike CAS Maxima that is used in STACK, SymPy doesn't allow to turn off axioms during testing equality of expressions. Turning off axioms like commutativity and associativity is helpful as one needs to determine if student's expression is in the certain form - expanded enough, etc. However, it turned out that it was much easier to embed SymPy than Maxima in a multi-user web-application. Therefore it was decided to use SymPy and develop extra heuristics for checking if student's expression is in the certain form.

III. IMPLEMENTATION OF INPUT ANALYSIS IN ITS "VOLGA"

Within ITS "Volga" student enters task's solution part via special interface element (Fig. 1). Task's solution part entered via this interface element is called student's step. The interface element includes a text input field and a region where visual representation of inputted formula is displayed. The visual representation of formula is interactively generated by MathJax [7]. Formula syntax is almost the same as it is in LaTeX.

Each possible task solution is represented with a set of instances of class "Milestone". Given which milestone corresponds to student's step, the correctness of student's step is usually determined by checking if the step formula is equivalent to the corresponding milestone formula. Equivalence of formulas is determined by SymPy "simplify" function.

All formulas are preprocessed before equivalence check. Since Mathjax syntax differs from SymPy syntax, and students are allowed to use simpler rules instead of some overly complicated LaTeX rules, preprocessing includes replacement of some symbol sequences by other symbol

Шаг 1

The screenshot shows a user interface with two buttons at the top: 'Проверить шаг' (Check step) and 'Подсказка по шагу' (Hint for step). Below the buttons is a text input field containing the formula: $p(a,b)=\sqrt{(a_1-b_1)^2+(a_2-b_2)^2+(a_3-b_3)^2+(a_4-b_4)^2}$. Below the input field, the same formula is displayed in a larger font, with a link 'переключить на легкий интерфейс' (Switch to easy interface) below it.

Figure 1. An example of student's step.

sequences (e.g. '^' by '**'). During preprocessing all task notations in formulas are replaced by corresponding symbols "x1y", ..., "xny" as well (when entering step formulas, student should only use notations from the certain list displayed by the user interface of ITS "Volga"). It allows not only to avoid situations in which presence of some notations (e.g. "p(a,b)", "x_a") causes exceptions during equivalence check, but to efficiently plug into the formula multiplication signs forgotten by the student.

We should also note that in some cases standard equivalence check procedure is conducted jointly with additional heuristics. Namely, if a part of milestone formula contains prefix "#almost#", an heuristic launches that checks if the corresponding part of student's step formula is expanded as much as the part of milestone formula by comparing counts of plus, minus, multiplication and pow signs. In case of presence of prefix "#fixed#", an heuristic launches that checks if the corresponding part of student's step formula is exactly the same as the part of milestone formula. It removes spaces from the formulas and compares formulas as strings. For example, if the milestone formula is "#fixed# $x + y = \text{\#almost\# } a + b$ ", then if student's step formula is

- $x + y = a + b$, student's step is correct,
- $y + x = a + b$, student's step is wrong,
- $x + y = b + a$, student's step is correct,
- $x + y = a + b + 1 - 1$, student's step is wrong.

We first faced with need to run additional heuristics after standard equivalence check procedure when we started to implement the procedure that checked if students' solutions were detailed enough. Such kind of analysis is important, because teachers may ask low achieving students to provide more detailed solutions, to prevent cheating. Thus there is a problem of matching student's step formula with milestone formulas. Let's consider the following example.

Task conditions: Find distance between vectors $a = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$

and $b = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$ of the Euclidean space.

One of possible task solutions:

$$p(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2} \quad (1)$$

$$p(a,b) = \sqrt{(1 - 5)^2 + (2 - 6)^2} \quad (2)$$

$$p(a,b) = \sqrt{(-4)^2 + (-4)^2} \quad (3)$$

$$p(a,b) = \sqrt{32} \quad (4)$$

Formulas 2-4 correspond to consecutive simplification of the vector distance formula with coordinates plugged in. Formulas 2-4 are pairwise equivalent for standard equivalence check procedure. Let's say a teacher wants students' solutions to include formula 2, and student's solution will only include formula 4, then it's impossible to check if student's solution is detailed enough because formulas 2 and 4 are undistinguishable without use of additional heuristics. To solve this problem we changed ITS "Volga" in a way described below.

For steps corresponding to consecutive simplification of the same expression only one milestone is created. If a teacher wants to impose conditions on student's solution specification then he adds formulas that should be included in student's solution as additional properties of the milestone ("necessary formulas"). In the case described above we have:

Milestone 1

Base formula: (1).

Milestone 2

Base formula: (2),

"Necessary" formulas: (2), (3), (4).

If student's formula is correct and corresponds to the milestone with a non-empty set of "necessary" formulas, then it's additionally checked whether student's formula is equivalent to one of these formulas in the strict sense (that is, prefix "#fixed#" is used during equivalence check). If one equivalent "necessary" formula is found, then it's stored in the system memory that student used this particular formula. Student's solution is considered to be detailed enough if all "necessary" formulas corresponding to the milestones of the closest system's solution were used by student. Closest system's solution is determined based on the number of coincidences between the set of solution's milestones and the set of distinct¹ milestones, corresponding to correct student's steps.

The milestone corresponding to student's step is automatically determined during equivalence check procedure. For that first it's checked whether student's step formula is equivalent to each of formulas corresponding to the milestones of possible task's solutions. If student's step formula is equivalent to some milestone formula, then this milestone is the milestone sought. If student's step formula is equivalent to several milestone formulas, then it's additionally checked whether it's equivalent to these formulas in the strict sense (that is, prefix "#fixed#" is used during equivalence check).

IV. THE PROBLEM OF MULTIPLE FORMULA COMBINATIONS GENERATED BY STUDENTS

The suggested procedure of checking students' solutions seems to have one critical shortcoming. Since many different

¹ By "distinct" we understand the option of SELECT operator of SQL-language

combinations of the same set of interrelated formulas can be generated by different students while entering a solution of any task, it is necessary to provide solutions including all the combinations. Providing so many solutions is tedious and labor-consuming. Let's consider the example of the task, where students need to find the distance between two vectors a, b of the Euclidean space.

To solve this task, students need to use the following formulas:

$$p(a, b) = |c|, \quad (5)$$

$$\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} a_1 - b_1 \\ a_2 - b_2 \end{pmatrix}, \quad (6)$$

$$|c| = \sqrt{(c, c)}, \quad (7)$$

$$(c, c) = c_1^2 + c_2^2 \quad (8)$$

Combination of formulas reduces to substitution of some variables in one formula by their values in the form of algebraic expressions derived from other formulas. For example, the combination of formulas (5) and (7) is:

$$p(a, b) = \sqrt{(c, c)}, \quad (9)$$

the combination of formulas (5), (7), (8) is:

$$p(a, b) = \sqrt{c_1^2 + c_2^2} \quad (10)$$

Let's build a graph that represents computability relations between formulas. Each vertex of the graph corresponds to one or several variables. Each edge corresponds to one formula. If there is an edge from a to b , then variable a can be derived from the formula that corresponds to the edge and includes only variable b . It's easy to see that the graph is transitive. For the sake of a clearer representation, not all edges are shown in Fig. 2.

If a new class "Combinatorial Milestone" that includes information about the graph representing computability relations is added to ITS data model, then students' solutions check procedure includes the following steps:

1. if there is a vertex of the graph that corresponds to the variables that are in the left part of the student formula, remember that vertex (let it be a),
2. if there is a vertex of the graph that corresponds to the variables that are in the right part of the student formula and this vertex is below vertex a , remember that vertex (let it be b),
3. form the "combinatorial milestone" formula by combining formulas moving along the way from vertex a to vertex b ,
4. apply standard equivalence check procedure that tests if student's formula is equivalent to the "combinatorial milestone" formula.

Let's consider the following example. Student entered the following formula:

$$|c| = c_1 + c_2 \quad (11)$$

"Combinatorial milestone" formula is formed as the combination of formulas (7) and (8) (these formulas

correspond to graph edges if we move from vertex corresponding to $|c|$ to vertex corresponding to c_1, c_2). "Combinatorial milestone" formula is:

$$|c| = \sqrt{c_1^2 + c_2^2} \quad (12)$$

From the standard equivalence check procedure it follows that (12) is not equivalent to (11), therefore student's step is incorrect.

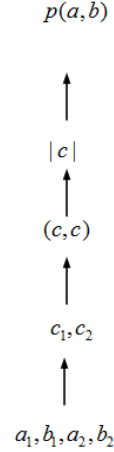


Figure 2. The graph that represents the computability relations between formulas in the task of computing distance between vectors of the Euclidean space.

The enhancement of this procedure is certainly the topic of the next articles. The goal of this section was to show that suggested solutions check procedure doesn't have critical shortcomings.

V. SUMMARY AND FUTURE WORK

Quality of the proposed procedure of checking students' solutions strongly depends on the capabilities of function "simplify" from SymPy library. The following question arises: how we can describe the class of expressions for which the function "simplify" always produces the result. As it was noted by developers of the library, this question cannot be yet accurately answered, because "simplify" function is an heuristic that they are constantly trying to improve. If the expression is really identical to 0, but doesn't simplify to 0, it is usually due to one of the following reasons:

1. the desired simplification is very complicated,
2. simplification is not applicable for some values of variables.

The default SymPy assumption is that all symbols are complex numbers, so "simplify" function does not use simplifications that are not applicable to all complex numbers. For example, expression

$$x = \sqrt{x^2} \quad (12)$$

is true only if x is positive number. SymPy developers note that this case can be avoided by additional configuration of simplifying functions, invoked during operation of "simplify" function.

These features of “simplify” function give hope that the solution of not too difficult tasks will be successfully checked by the procedure described above. The procedure of checking students’ solutions was successfully tested on the solutions of students of the Psychology Department of Moscow State University attending the course of linear algebra. Within the expansion of ITS to other subject areas it may be necessary to revise not only the suggested procedure but SymPy library as well.

REFERENCES

- [1] SymPy Development Team.. “SymPy: Python library for symbolic mathematics.” URL: <http://www.sympy.org>
- [2] Vasilyev S. (et al.) “Adaptive Approach to Developing Advanced Distributed E-learning Management System for Manufacturing”, Preprints of the 13th IFAC Symposium on Information Control Problems in Manufacturing (FR-C86). Moscow, 2009, pp. 2198-2203.
- [3] E. Melis, J.Siekmann. “ActiveMath: An Intelligent Tutoring System for Mathematics”. Artificial Intelligence and Soft Computing (ICAISC 2004). Springer Berlin Heidelberg, 2004. Vol. 3070. pp. 91-101.
- [4] R. Bradford, J.H. Davenport, C. Sangwin. “A Comparison of Equality in Computer Algebra and Correctness in Mathematical Pedagogy”. International Journal for Technology in Mathematics Education, 2009. Vol.16, № 1.
- [5] VanLehn K. (et al.) “The Andes Physics Tutoring System: Lessons Learned.” International Journal of Artificial Intelligence in Education”. 2005. Vol . 15, № 3. pp. 147-204.
- [6] J.A. Shapiro. “An Algebra Subsystem for Diagnosing Students’ Input in a Physics Tutoring System” . International Journal of Artificial Intelligence in Education (IJAIED), 2005. № 15. pp. 205-228.
- [7] Mathjax Development Team. “MathJax: open source JavaScript display engine for mathematics”. URL: <http://www.mathjax.org/>